Progress Review 10
Project Pegasus

Tushar Agrawal

Team A – Avengers Ultron
Teammates: Pratik Chatrath, Sean Bryan

ILR #9
March 17, 2016

1. Individual Progress

After the last Progress Review, the UAV was capable of autonomously following the visual marker. It could also communicate its odometry to the Navigation stack and receive velocity commands.

Autonomous Takeoff and Landing and Behavior Routine
I started working on the final frontier to fully autonomous UAV, which was Autonomous Takeoff and Landing. As we had shifted to PX4 due to issues in the APM stack (discussed later), I set up the PX4 simulator (Image 1) and tested a sample code for autonomous motion. It gave results as expected. I increased the height in the autonomous way points for takeoff and reduced the height for landing. This worked as expected on the simulator.

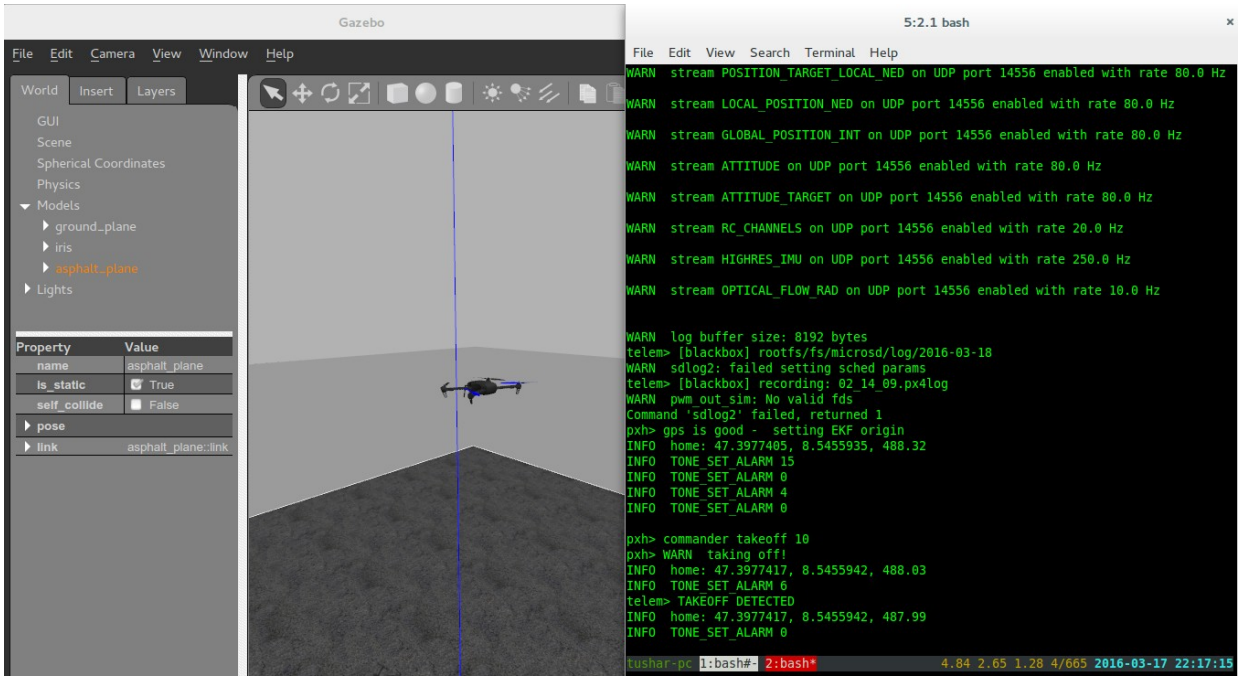

Image 1. PX4 Flight Simulator on Gazebo. UAV is currently at position (0,0,10m)

While testing the same on the real UAV, I realized that setting a way point at (0,0,10) for take off to 10m was not the ideal solution, as the UAV might be taking off from a new location which was not (x=0, y=0) and would try to move sideways before gaining altitude, which may topple it over. I substituted it with a take off at current location functionality, using which the UAV took off at the current position itself. Even while landing, the UAV's altitude sensor was not reliable, and even with the recommended z=-0.2m final height, the UAV would not reach the ground. To fix that, I re-calibrated the altitude sensors at the flight area (Flagstaff Hill), and allowed it to descend down to z = -1m to fully touch the ground. This resulted in a swift takeoff and a complete landing. The results can be seen in video 1, shared later in this section.

Furthermore, I made the framework for the behavior module which controls the high-level activity of the UAV during the delivery mission. This was designed as a basic state machine as shown in Image 2.

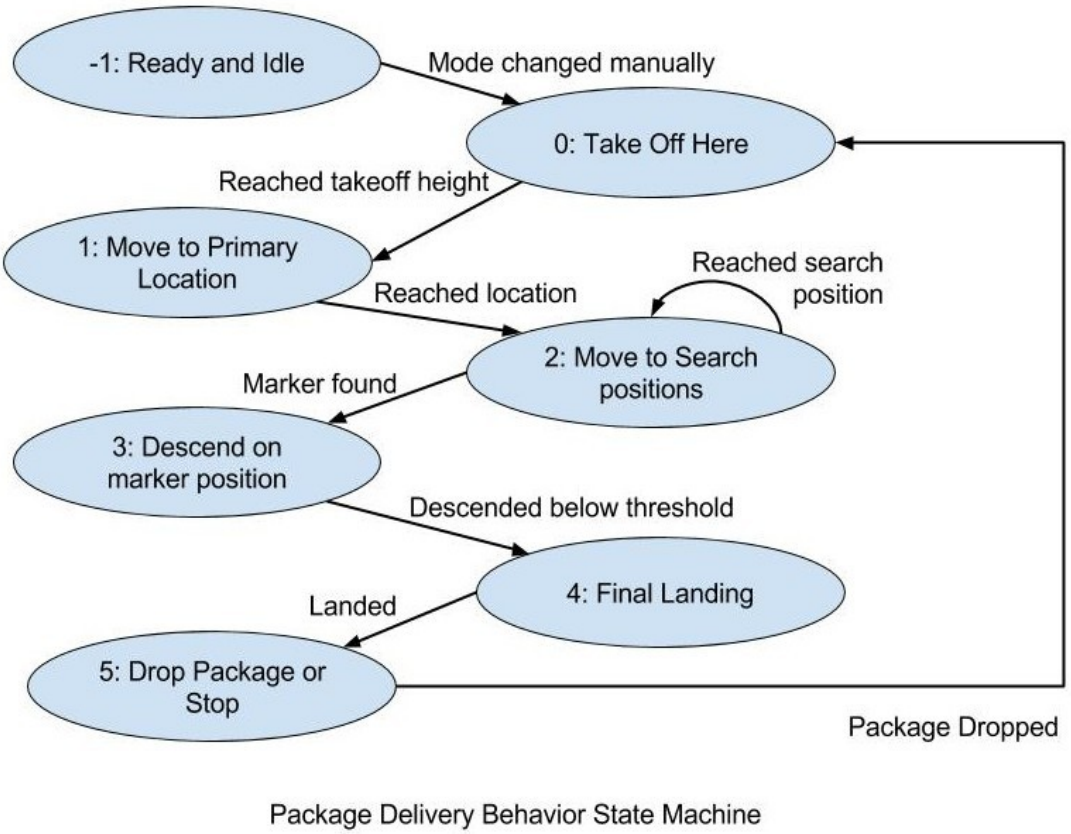

Image 2: Bahavior State Machine
Note: This is just the initial implementation, many cases areyet to be encoded

Functionalities are activated based on the state of system:

1. **State -1: Idle.**
2. **State 0: Take off from current location.**
   - Blocking until height is achieved.
   - Does not look for marker.
3. **State 1: Move to house**
   - Blocking on reaching set position
   - Does not look for marker
4. **State 2: Search for marker**
   - Blocking on individual position goals
   - Looks for marker.
5. **State 3: Marker found. Descend.**
   - Keeps a lookout for the marker
   - Continues descent when the marker is visible, until it reaches a set height
   - Individual position are not blocking. They may be overwritten by the marker detection system
6. **State 4: Final Descend/Landing.**
   - Remembers the filtered marker location and continues descend on that position.
7. **State 5: Drop package.**
   - Initiates drop off of the package (Yet to be implemented)

This state machine was used to run the entire process in a local space without obstacle detection. The results are shown in this video: https://www.youtube.com/watch?v=Jz0qRndWkbM (Video 1). A simple landmower search pattern was used as a proof of concept (Illustrated in Image 3). We were able to land around 60 cm from the center of the marker, well inside our requirements of 2 meters.

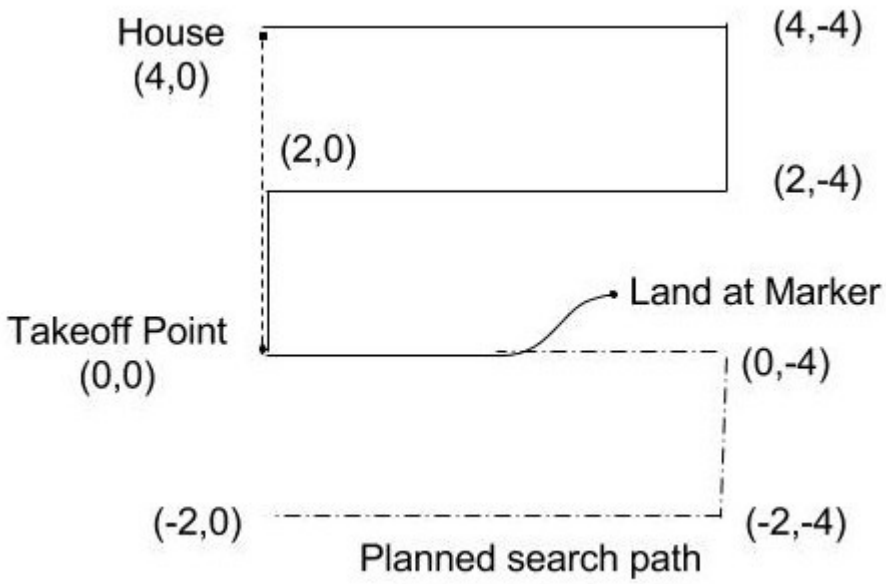

Image 3. Trajectory planned and executed by UAV in video 1. Proof of concept for behavior system

Obstacle Avoidance

In parallel, I worked on setting up the Navigation Stack on the Odroid. The navigation stack had been verified by Pratik by multiple tests on the Laptop.

Also, during our last fight, I had noticed that the velocity commands used by the UAV had to be given in the global frame (Based on position and direction of the UAV when powered up). But the navigation stack was producing these in the local frame of the robot. I wrote a converter, very similar to the marker position frame converter to convert the local velocities to global ones, and tested it briefly on the PX4 simulator.

In addition, the navigation stack used odometry from the UAV. Surprisingly, this was made available in the latest (0.17.0) version of mavros, having been added only 2 weeks earlier. I upgraded mavros on the Odroid, and after resolving few issues, was able to get the odometry information from the pixhawk.

We tested the autonomous A to B navigation with obstacle avoidance on the real UAV. A goal of (4,0) was sent to the navigation stack, which produced corresponding velocity commands. These were converted into the global frame and forwarded to the UAV through mavros. After many iterations for tuning the parameters in the navigation stack, we were able to get good response for avoiding obstacles from the UAV. A video of the same can be seen here, https://www.youtube.com/watch?v=r1X6UWgTfsM (Video 2).

Odroid control of the NicaDrone (Electro-Permanent Magnet)
I wrote a basic version of the code required to control the Nicadrone through the Odroid. This used the Mavros DO_SET_SERVO command. Unfortunately, we switched to PX4 due to other major issues on the obstacle avoidance front, and that functionality was not exposed in the same way on the PX4 firmware. Later, I figured out a way to implement the required functionality, but it was very time consuming. Hence, we decided to de-prioritize this for PR10.

2. Challenges

Many challenges were faced during this process:

- As APM firmware would not be able to take angular velocities, we had to shift to the PX4 firmware, which affected our timelines initially.
- The PX4 simulator took significant effort, as it required the firmware to be built. It finally worked after a suspicious 'make clean' command.
- Final landing required the UAV to descend down to -1m in Z to finally land.
- Some control issues and altitude issues on the PX4 required re-calibration at the flying area (Flagstaff Hill)
- The UAV suffered a major crash few days before the progress review. Thankfully, we were able to make fast repairs and have it up in the air the very next day (and shoot the videos mentioned here)

Even with these challenges, we were able to meet the prioritized goals of the Progress Review.

Current challenges include the control of NicaDrone through the Odorid and marker search while avoiding obstacles.

3. Teamwork

Based on our test plans, Pratik was going to work on the obstacle avoidance, Sean on the Nicadrone control through the Odroid and I was going to work on the Autonomous Landing.

Pratik and I worked on most of the Obstacle Avoidance set up and testing together. We did various tests and appropriately tweaked the parameters to get a quick and safe response for avoiding obstacles.

Sean set up his system for learning mavros and working on the odroid control of nicadrone next. I helped him set up the QgroundControl GUI.

After the crash, we all worked together to repair the UAV and get it capable of flying as soon as possible. We also sat together and discussed the design for the obstacles to be used for testing and SVE demonstration.

4. Future Work

Working with the test plan in mind, Pratik and I shall concentrate on integrating the obstacle avoidance system to continue through multiple waypoints to enable marker search, and possibly tweak the parameters further for optimal response. Sean will work initially on building the test environment, including obstacles and landing bases. In

addition, he and I will also work on getting the Nicadrone working off the Odroid.

By the end of this Progress Review we hope to at least have a system which can deliver packages without any obstacles in its path, and separately, can avoid obstacles in a basic path. Integration from there should be relatively less complicated.